Introduction à la sécurité – Cours 3
Contrôle d’accès

Catalin Dima

Contrôle d’accès

- Une des bases de la sécurité :
  - On permet à une action d’être exécutée que si elle respecte une certaine politique.
  - Permet de renforcer la *confidentialité* ou l’*integrité*.
- Contrôle d’accès *discrétionnaire* – basé sur l’identité et la possesion.
  - UNIX/Linux.
- Contrôle d’accès *obligatoire* – basé sur l’existence d’un Serveur de Sécurité.
  - Modèles basés sur les rôles, sur les organisations, sur les lattices de sécurité...
  - Renforcement de types.
  - Modèles intéressants théoriquement : Take-grant, protection schématique.

Contrôle d’accès discrétionnaire

- Basé sur l’identité :
  - *Utilisateurs* – les sujets des actions.
  - *Objets*.
  - Chaque utilisateur *possède* un certain ensemble d’objets, avec lesquels il peut faire ce qu’il veut.
- *Avantages* : décisions decentralisées, surcoût faible de déploiement, tolérance aux attaques de type déni de service.
- *Désavantages* : difficile d’analyser, peut engendrer des failles de sécurité, peut ne pas être utilisable pour l’implémentation de certains principes de sécurité.

Contrôle d’accès obligatoire

- *Serveur de sécurité* (SS), censé renforcer une certaine politique.
  - Exemple : politique “militaire”, avec des niveaux de sécurité.
- Règles d’accès, basées sur des *permissions/droits*.
- Toute action doit être validée par le SS,
- ... qui prend sa décision en consultant l’ensemble des règles.
- *Avantages* : conception et analyse unitaire, modèles complets permettant l’implémentation d’une majorité des principes de sécurité.
- *Désavantage* : surcharge importante lors de la gestion de grands systèmes, possible concentrateur d’attaques, tolérance aux fautes réduite.

Modèle HRU (Harrison-Ruzzo-Ullman)

- Composants de l'état de protection :
  - Sujets S : utilisateurs, groupes, processus, services...
  - Objets O : fichiers, processus, services...
    À noter : $S \subseteq O$.
  - Actions Act : lecture, exécution, communication entre processus, création.
  - Droits R : chaque action implique l'existence des droits entre les participants à l'action !
  - Correspondance actions - droits.
  - **Matrice de droits d'accès** $M : S \times O \longrightarrow 2^R$.
- État de protection du système = $\big(S, O, M\big)$.
  - Image instantanée des caractéristiques du système.

Modèle HRU

- Chaque action est “initiée” par un sujet,
- ...et implique un certain nombre de participants (sujets ou objets).
- On aura donc les participants de l’action $Part : Act \longrightarrow \mathbb{N}$.
  - Exemple d’action : $videoconf$, $Part(videoconf) = 4$.
    - Quatre participants, dont un qui est l’initiateur.
    - On écrira alors $videoconf(Alice, Bob, Charlie, Danny)$, avec $Alice$ comme initiateur.
  - Action $fork$, $Part(fork) = 2$.
    - Exemple d’action : $fork(33175, 35881)$;
    - Le sujet de PID 35881 est créé !
  - Action $kill(33175, 35881)$.

Actions et permissions

- Chaque action peut être autorisée si certaines permissions sont satisfaites par les participants.
- Action act associée à une contrainte C sur la matrice des droits d'accès.
- Contrainte spécifiée comme combinaison booléenne des permissions élémentaires
  - $r \in [x, y]$: le sujet x a le droit r sur l'objet y.
- La sémantique des permissions est donnée par cette affectation.
- Exemples :
  - Pour $videoconf(x, y, z, t)$:

$$Perm(videoconf) = \text{if } v \in [x, y] \text{ and } v \in [y, x] \text{ and } \ldots$$

  - $Perm(kill(33175, 35881))$ devrait spécifier le fait que les 2 processus appartiennent au même utilisateur (comment écrire ça ?)
  - Certaines actions peuvent demander plus de permissions pour la même paire sujet/objet (ou sujet/sujet) – e.g. lancement en exécution.

Actions et matrice d'accès

- Chaque action pourrait avoir aussi une influence sur la matrice de contrôle d'accès !
  - `chown,chmod,...`
- Dans ces cas, il faut aussi modéliser cet effet.
- On définit alors les **opérations primitives** qui peuvent apparaître en tant qu'effet des actions sur M :
  - enter r into $[X, Y]$.
  - delete r from $[X, Y]$.
  - create subject X.
  - create object X.
  - destroy subject X.
  - destroy object X.
- Attention, ces sont des *prototypes* ! X, Y sont des *variables*, seulement r est instancié !

Sémantique des opérations

- *Configuration* (instantanée) : (S, O, M), ou $M : S \times O \longrightarrow R$ **et** $S \subseteq O$.
- *Transition* par une opération primitive $(S, O, M) \vdash_{op} (S', O', M')$

1. Si $op =$ enter r into $[X, Y]$, alors

$$S' = S, O' = O, M'(a, b) = \begin{cases} M(X, Y) \cup \{r\} & \text{si } a = X, b = Y \\ M(a, b) & \text{sinon} \end{cases}$$

2. Si $op =$ delete r into $[X, Y]$, alors

$$S' = S, O' = O, M'(a, b) = \begin{cases} M(X, Y) \setminus \{r\} & \text{si } a = X, b = Y \\ M(a, b) & \text{sinon} \end{cases}$$

3. Si $op =$ create subject X alors

$$S' = S \cup \{X\}, O' = O \cup \{X\}, M'(a, b) = \begin{cases} M(a, b) & \text{si } a \in S, b \in O \\ \emptyset & \text{sinon.} \end{cases}$$

4. Si $op =$ destroy subject X alors

$$S' = S \setminus \{X\}, O' = O \setminus \{X\}, M'(a, b) = M(a, b), \text{ pour tout } a \in S', b \in O'$$

Actions & matrice d'accès

- *Commande* = action qui peut avoir une influence sur la matrice de contrôle d'accès.
- *Transition* par une commande α, $(S, O, M) \vdash_\alpha (S', O', M')$

  command $\alpha(X_1, \ldots, X_n)$
  if r_1 in $[X_{i_1}, X_{j_1}]$ and $\ldots$ and r_k in $[X_{i_k}, X_{j_k}]$
  then
  op_1
  $\ldots$
  op_p

- Sémantique :

  1. Vérification des pré-conditions de la commande :

  $$r_t \in M(X_{i_t}, X_{j_t}) \text{ pour tout } 1 \leq t \leq k$$

  2. Exécution des opérations primitives dans l'ordre donné :

  $$(S, O, M) \vdash_{op_1} (S_1, O_1, M_1) \vdash \ldots \vdash_{op_p} (S_p, O_p, M_p) = (S', O', M')$$

- On dénote alors $(S, O, M) \longrightarrow (S', O', M')$.
- Les opérations sont ordonnées à l'intérieur d'une commande, *mais les commandes ne sont pas ordonnées !*

Exemple : système hiérarchique

- Supposons un système avec des niveaux de documents (public, classifié, top secret).
- On veut pas que le grand public lise des documents secrets.
- Mais on ne veut pas non plus d'avoir des "taupes" !
- Politique Bell-LaPadula (sécurité multiniveaux – on va revenir là-dessus) :
  - No read-up, no write-down.
- Implémentable dans le modèle HRU :
  - *Niveaux de sécurité* : `P, C, TS`.
  - Éventuellement des permissions `r,w` – mais pas nécessaire !
  - Tous les objets sont sujets aussi et possèdent un niveau de sécurité !
  - Actions : $read, write$.

$$Perm(read(X,Y)) = \begin{array}{l} \text{if } ((\texttt{P} \in [X,X]) \text{ and } (\texttt{P} \in [Y,Y])) \\ \text{or } ((\texttt{C} \in [X,X]) \text{ and } (\texttt{P} \in [Y,Y])) \\ \text{or } \ldots. \end{array}$$

- Problème : est-ce qu'on peut modifier les niveaux de sécurité des sujets/objets *sans toucher aux principes no read-up/no write-down ?*

Exemple 2 : UNIX (simplifié et modifié)

- Attribut propriétaire : `own`.
- Fichiers = sujets de type spécial.
- Privilèges de propriétaire : `oread, owrite, oexec`.
  - Si f a X comme propriétaire et peut être lu par tous,
  - ... alors on a `aread` $\in M(f, f)$ et `own` $\in M(X, f)$.
- Donc action $read(X, Y)$ avec

$$Perm(read(X, Y)) = \text{if } (\texttt{aread} \in M(Y, Y)) or ((\texttt{oread} \in M(X, Y)) and (\texttt{own} \in M(X, Y)))$$

- Les autres permissions `awrite, aexec` – définies de manière similaire.
- `chmod` : *commande*, car modifie l'état de sécurité :

command `chmod_read_all`(X, Y)
  if `own` $\in [X, Y]$
  then enter `oread` into $[Y, Y]$

Politiques de sécurité et contrôle d'accès

On fixe un système (S, O, M) et un ensemble d'actions (et commandes) C.

- Configuration atteignable = (S', O', M') telle qu'il existe

$$(S, O, M) \longrightarrow (S_1, O_1, M_1) \longrightarrow \ldots \longrightarrow (S_p, O_p, M_p) = (S', O', M')$$

- Un système de contrôle d'accès peut **provoquer la fuite d'un droit** r s'il existe une configuration atteignable (S', O', M') telle que $r \in M'(x, y)$ pour certains $x \in S, y \in O$.
- Un système d'accès est **sûr par rapport à un triplet** $(x, y, r) \in S \times O \times R$ si dans aucune configuration atteignable (S', O', M') x n'a le droit r sur y ($r \notin M(x, y)$).
  - Notion *locale*.
- Un système d'accès est **sûr par rapport à un droit** $r \in R$ si dans toute configuration atteignable (S', O', M') on ne peut jamais trouver x, y tels que $r \in M'(x, y)$.
  - Notion *globale*.
- Politique de sécurité = ensemble de propriétés de sûreté.

Décidabilité

- Est-ce que le système de transitions est (en soi-même) un mécanisme renforçant une politique de sûreté ?
- *Question équivalente* : est-ce qu'on peut construire un algorithme tester la non-fuite d'un droit pour tout système de type matrice d'accès ?
- Si le système ne permettrait pas de créer de nouveaux sujets/objets, alors simulation jusqu'à la couverture de toutes les configurations possibles.
- *Problème* : création de sujets et/ou objets !
  - L'espace de configurations possibles peut être infini !
  - La procédure consistant a construir toutes les configurations atteignables ne s'arrêterait pas en général !
  - Exemple : UNIX et création de nouveaux processus.

Décidabilité

Théorème (Harrison, Ruzzo, Ullman, ’76) : Le problème de sûreté pour les systèmes de type matrice d’accès est *indécidable*.

- Idée : on peut coder le comportement d’une machine de Turing en tant que système de contrôle d’accès
  - Sujets = cases sur la bande.
  - Droits = états de la machine + symboles de bande + droit spécial symbolisant la relation d’adjacence.
  - Position de la tête de lecture = un certain sujet (case) possède le droit q (état courant) sur lui-même, et tous les autres sujets n’ont pas de droit sur eux-mêmes.
  - Transitions dans le système simulent les transitions dans la machine de Turing.
  - On peut faire même sans suppression de droits !
- Mais cela n’empêche pas d’avoir de sous-classes ou le problème soit décidable.

Take-grant

- Droits : *t* (“take”) et *g* (“grant”).
- Commandes de type spécial :

$$\begin{aligned}
\mathbf{take}(x,y,z) &: \text{if } t \in [x,y] \text{ and } \alpha \in [y,z] \\
&\quad \text{then enter } \alpha \in [x,z] \\
\mathbf{grant}(x,y,z) &: \text{if } g \in [x,y] \text{ and } \alpha \in [x,z] \\
&\quad \text{then enter } \alpha \in [y,z] \\
\mathbf{create}(x,y) &: \text{ create } x; \\
&\quad \text{enter } t, g \in [x,y]
\end{aligned}$$

- “take” permet à x de récupérer tous les droits que y possède sur z ;
- “grant” permet à x de propager à y ses droits sur z.
- Tout objet crée par x est possédé par celui-ci, qui peut donc récupérer ou lui transmettre tous les droits.

Décidabilité

Théorème : Il existe un algorithme permettant de décider, pour tout système de type take-grant, si le système provoque la fuite d'un droit.

- Chemin *tg* : chemin dans le graphe représentant la matrice de droits d'accès contenant que des arêtes t ou g.
  - Orientation d'arêtes non-importante.
- **Théorème :** En supposant $S = O$, pour $x, y \in S$, x peut récupérer le droit α sur y si et seulement si il existe un $z \in S$ tel que $\alpha \in [z, y]$ et x et z sont reliés par un chemin *tg*.

Systèmes typés

TAM = *Typed Access Matrix* :

- Primitives : S = sujets, O = objets, R = droits,
- T = **types** et $t : O \longrightarrow T$: pour tout objet $o \in O$, $t(o)$ donne son type.
- Commandes *typées* :

$$\begin{array}{l}\text{command nom}(X_1 : t_1, \ldots, X_n : t_n) \\ \qquad \text{if } r_1 \text{ in } [X_{i_1}, X_{j_1}] \text{ and} \ldots \text{ and } r_k \text{ in } [X_{i_k}, X_{j_k}] \\ \qquad \text{then } op_1; \ldots; op_p\end{array}$$

- Sémantique : les substitutions des variables doivent respecter les types.

Systèmes typés

- *Monotonicité* : on ne dispose que de commandes de création de sujets ou de rajout de droits (pas de destruction/elimination de droits).
- *Canonicité* : les commandes de création sont inconditionnelles.
- **Théorème :** tout système TAM monotonique (MTAM) est équivalent à un système MTAM à commandes canoniques.
  - Équivalence : on peut “simuler” le système canonique dans le système initial et vice-versa.
- Relation de *dépendance des types* : si dans une commande on a une variable d’entrée $x : t$ et une variable de sortie $y : t'$ (c’est à dire, y est créé par cette commande !), alors $t' \prec t$.
  - Peut être étendue à une *relation de préordre* par reflexivité et transitivité.
  - En général l’extension *n’est pas un ordre partiel* : cycles de dépendance.

Implémentation : ACL et capabilités

- Acces Control Lists : $L : O \longrightarrow [S \times 2^R]$
  - Colonnes de la matrice des droits d'accès.
  - Langage de spécification dépendant du système.
- Capabilites : $C : S \longrightarrow [O \times 2^R]$
  - Lignes de la matrice des droits d'accès.
- Pourquoi opter pour l'une ou l'autre :
  - Si $nbobjets >> nbsujets$ alors ACL.
  - Capabilités utiles quand on veut enregistrer le comportement des sujets.